VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvovaní individuální odborné praxe
Individual Professional Practice in the Company

2016 Richard Vašek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Richard Vašek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Tieto Czech s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Mgr. Marek Menšík, Ph.D.**

Konzultant bakalářské práce: Mgr. Zdeněk Dřizga

Datum zadání: 01.09.2015

Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2016

Vašek

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 20. dubna2016

.....

Tieto Czech s.r.o.
28. října 3346/91
702 00 Ostrava - Moravská Ostrava
IČO 64608051 DIČ CZ64608051

Rád bych na tomto místě poděkoval firmě Tieto Czech s.r.o. za umožnění vykonání bakalářské praxe. Zejména bych chtěl vyjádřit poděkování Mrg. Zdeňku Dřizgovi, který mě po dobu mé bakalářské praxe vedl, dále bych chtěl poděkovat Ing. Janu Žídkovi, který mi často pomáhal s architekturou Karellenu. Nemalá poděkování patří i dalším kolegům, kteří mi byli v případě nutnosti schopni pomoct se spoustou věcí.

Abstrakt

Tato Bakalářská práce popisuje mé působení v Tieto Czech, kde jsem vyvíjel systém pro velkou severskou společnost a interní projekty, které používá celá pobočka českého Tieta. Práce ve svém úvodu popisuje firmu a poté jednotlivé úkoly, jenž jsem, ve firmě dostal a plnil, dále jsou popsány některé technologie a strategie, které jsem použil k vyřešení jednotlivých úkolů. V závěru se zmiňuji o zkušenostech a vědomostech, které jsem během praxe nabyl a shrnutí praxe jako takové.

Klíčová slova: Bakalářská práce, Tieto, Java, Spring, Git, JavaScript, web, monitoring

Abstract

This Bachelor thesis describes my activities in Tieto Czech, where I developed a system for large Nordic company as well as internal projects that are used by everyone in Tieto Czech. In the introduction of this thesis I describe the company and then tasks that were assign to me, then I describe some of the technologies and strategies that I have used and learned. In conclusion, I mention the experience and knowledge that I gained during my practice in the company and a summary of practice as such.

Key words: Bachelor thesis, Tieto, Java, Spring, Git, JavaScript, web, monitoring

Obsah

Seznam použitých zkratek a symbolů

CSS – Cascading Style Sheets
ES6 – Standart Ecma Script verze 6
GIT – Distribuovaný systém správy verzí
Java EE (J2EE) – Java Platform, Enterprise Edition
Jboss – Aplikační server pro J2EE 6
Jenkins – Task runner
JIRA – Softwarový nastroj pro řízení projektů
JPA – Java Persistence Api
JSF – Java Servlet Faces
LDAP – Lightweight Directory Access Protocol
Maven – Build manager pro Javu od Apache
Primefaces – Rozšíření standartu JSF o nové komponenty
Spring – Java framework
Thymeleaf – Templatovací jazyk pro generování HTML
Tomcat – Malý Java server
Wildfly - Aplikační server pro J2EE 7

Seznam Obrázků

Seznam útržků zdrojového kódu

1 Úvod

Při svém výběru bakalářské praxe se mi naskytla možnost absolvovat odbornou praxi ve firmě. Odbornou praxi jsem se rozhodl absolvovat ve firmě Tieto Czech a po svém úspěšném pohovoru jsem do firmy nastoupil na začátku 2. ročníku svého bakalářského studia, kdy jsem začal plánovat své vykonání odborné praxe.

Očekával jsem, že konečně poznám, jak vypadá práce v týmu v IT firmě, jelikož jsem do té doby neměl o práci v IT žádnou představu. Dále jsem chtěl získat zkušenosti z dlouhodobějšího vývoje větších systému v týmu lidí a chtěl jsem mít možnost porovnat akademické prostředí s prostředím firemním.

V druhé části této bakalářské práci bych chtěl uvést popis firmy, mou pozici ve firmě. Na začátku třetí části poté uvádím úkoly, jenž mi byly na praxi uděleny. Dále pak rozebírám jednotlivé části úkolů detailněji. Na konci pak uvádím své poznatky z práce na jednotlivých projektech a technologie použité při vypracovávání úkolů. V závěru práce shrnuji své nabyté zkušenosti na odborné praxi a zhodnocení mé praxe a mých očekávání.

2 Odborné zaměření firmy a popis pracovního pozice

2.1 Odborné zaměření firmy

Tieto je největším dodavatelem IT služeb pro soukromý i veřejný sektor ve Skandinávii. Jako důvěryhodný partner v transformaci IT pomáháme našim zákazníkům přizpůsobit jejich procesy a podnikání požadavkům trhu. V Tieto klademe velký důraz na profesní růst a výsledky.

Společnost Tieto, založená v roce 1968, se sídlem v Helsinkách a čistými tržbami 1,8 miliard EUR zaměstnává na 15 000 expertů a působí ve více než 20 zemích světa. Její akcie jsou obchodovány na burze NASDAQ OMX v Helsinkách a Stockholmu.

Tieto Czech s.r.o.

Do České republiky společnost Tieto vstoupila v roce 2001 a v roce 2004 otevřela své softwarové centrum v Ostravě. S více než 2 000 zaměstnanci je jedním z největších zaměstnavatelů v oblasti IT služeb v České republice a největším v Moravskoslezském kraji. Z hlediska počtu kmenových zaměstnanců je česká pobočka třetí největší pobočkou Tieto korporace na světě. První dvě místa zaujímají mateřské země Finsko a Švédsko.

Nové sídlo společnosti Tieto Czech - Ostrava Tieto Towers

Během relativně krátké doby se společnosti Tieto podařilo vybudovat stabilní zázemí pro lidi různých věkových kategorií z různých koutů nejen České republiky, ale i celého světa. Stavba architektonicky pozoruhodné budovy, jejímž autorem je architekt Václav Hlaváček, začala v květnu roku 2011.

Koncem roku 2012 se zaměstnanci společnosti Tieto, kteří doposud v Ostravě pracovali ve čtyřech různých budovách, přesídlili do těchto nových prostor. Společnost věří, že tyto moderní, flexibilní a účelné kanceláře pomohou většímu komfortu jejích zaměstnanců. [4]

2.2 Pracovní pozice

Ve firmě jsem byl na pozici Software Developer. Mé zaměření se týkalo převážně vývoje v jazyce Java a populárním frameworku Spring a postupem času jsem se dostal k vývoji v jazyce Java Script a jeho nové verzi ECMAScript 2015. Dozvěděl jsem se o spoustě technologií, jako jsou NOSQL, ReactJS, AngularJS. Dále jsem se naučil používat verzovací systém GIT a systém pro správu úkolů JIRA. Měl jsem také možnost vyzkoušet si vedení týmu stážistů na několika projektech.

3 Zadané úkoly a časová náročnost

Na své praxi jsem dostal postupně 2 úkoly, na kterých jsem pracoval. První z nich se týkal vývoje monitorovacího systému s názvem Karellen. Druhý byl převzetí několika již rozběhlých projektů a vést tým vývojářů složený ze stážistů za účelem dalšího rozvoje těchto aplikací. Třetím úkolem bylo začít vyvíjet aplikaci založenou na JavaScript technologii.

3.1 Časová náročnost

První část mé praxe jsem strávil na vývoji monitorovacího systému s názvem Karellen. Tento úkol mi zabral větší část mého času na praxi a to 50 dnů, během kterých jsem pracoval na následujících úkolech:

> Kolektory, Konfigurátor, Dashboard architektura, Dashboard design, Status List Widget, Traffic Light Widget, Counter Widget, Režim celé obrazovky, Noční režim, Přidání jednotky do Entity, Export a dodatečné zpracování dat z databáze pro reporty, Teplotní mapa vč. zobrazování chyb, Nelineární barvy v teplotní mapě, Hlavní stránka dashboardu, Studie a implementace zabezpečení v mikroservice architektuře, Login modul s API pro přihlášení, LDAP implementace login API, Statická implementace login API, Automatizace transpilace ES6 do ES5, Odhlášení a ošetření vypršení session, Skupiny entit a uživatelů, Průzkum grafových databází a jejich použití za pomocí Spring FW, Oblíbené dashboardy, CSRF ochrana, Stránka se záznamy změn na projektu, Responzivita, Informační karty s grafem, Uživatelská práva na dashboardy, Sdílení dashboardů, Import naměřených dat v CSV formátu do databáze

Ve své druhé části praxe jsem pracoval na několika interních projektech, kde jsem se také staral o několik nových stážistů a pomáhal jim s jejich prací na projektech. Tato část mi na praxi zabrala 16 dní a po praxi v ní nadále pokračuji.

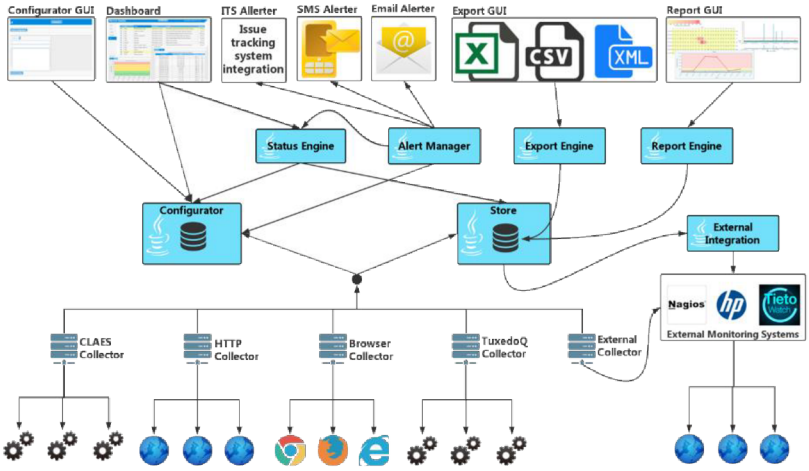
3.2 Monitorovací systém Karellen

Na tomto projektu jsem začal pracovat po nástupu do firmy. Na projektu jsem pracoval od jeho úplného začátku. Bylo vyžadováno rozvrhnout architekturu a zvolit správné technologie a následně vytvořit funkční aplikaci. Na projektu jsem po většinu času pracoval se dvěma dalšími stážisty a projekt byl v některých situacích konzultován s různými Architekty a Senior Developery. Při návrhu celé architektury systému jsem vycházel z literatury [1, 2, 5], která mi pomohla s návrhovými vzory a použitím Spring Frameworku.

3.2.1 Architektura systému

Systém jsme se snažili navrhnout velmi modulárně, proto se zvolilo použití tzv. Microservices. Systém je tedy rozdělen na několik dostatečně malých modulů, které spolu komunikují přes jejich REST API. Dále byla použita relační databáze, která byla rozdělena do dvou, ke kterým moduly přistupovaly. První databáze sloužila jako sklad naměřených dat a druhá jako sklad konfigurací systému.

Moduly sbírající data používaly svou vnitřní relační databázi, která byla ukládána jako soubor na disk a sloužila jako mezipaměť v případě nedostupnosti modulu pro ukládání dat do centrálního skladu dat.



Obrázek 1 Celková architektura Karellenu

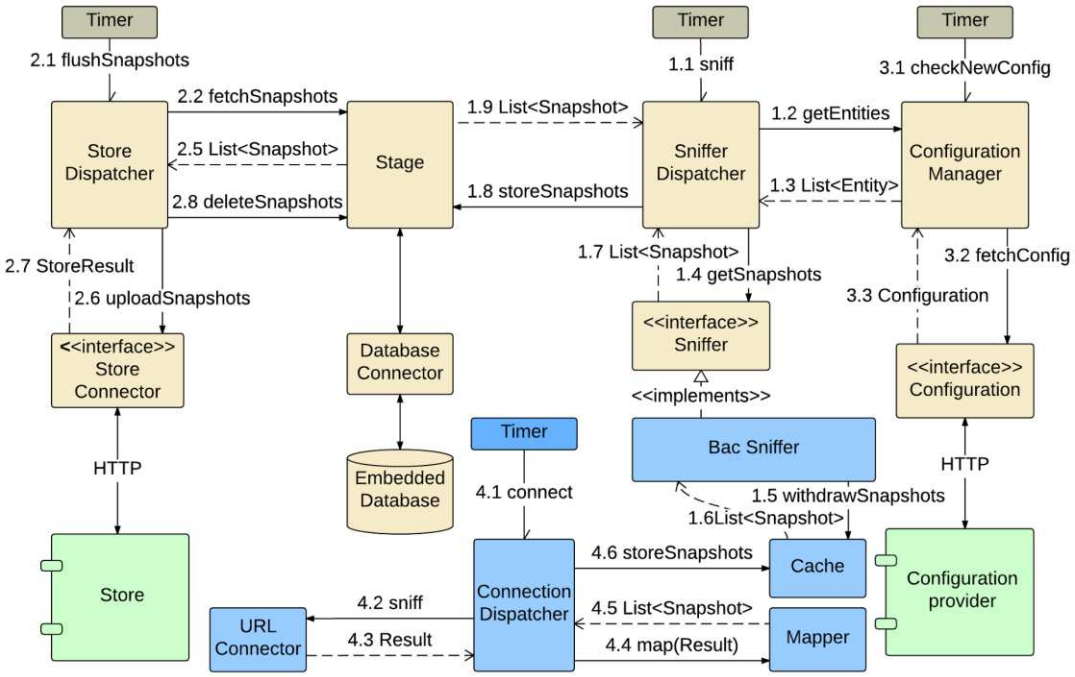
3.2.2 Sběr dat

Moduly pro sběr dat, slouží k monitorování různých systémů. Modul se skládá ze dvou částí: Collector (béžová barva) a Sniffer (modrá barva) a komunikuje s dvěma dalšími moduly a to jsou Store a Configurator (zelená barva). Modul jsem vytvořil tak, aby bylo velmi jednoduché napsat nový Sniffer přímo na míru požadavků na měření.

Modul je tedy vždy složen ze společné části nazvané Collector, ve které je obsažena byznys logika, která si zjistí, co se má měřit a jak často. Dále obsahuje mezipaměť ve formě vnitřní database a následný mechanismus odesílání dat do Store modulu pomocí REST API. Collector získává svou konfiguraci z REST API modulu Konfigurátor. Pro rozlišení konfigurací pro různé Collectory se používá ID, které je unikátní pro každý spuštěný Collector. Collector tedy po svém spuštění vytvoří požadavek na Konfigurátor se svým ID a dostane zpět svou konfiguraci, kterou zpracuje a začne s měřením. Poté v 30 sekundovém intervalu dělá další požadavek o novou konfiguraci, kde zasílá své ID a časovou značku, kterou dostal s poslední konfigurací. Konfigurátor porovná časovou značku od Collectoru a časovou značku uloženou u konfigurace a pokud se liší, pošle zpět aktuální konfiguraci z databáze, pokud jsou datové značky shodné, posílá zpět prázdnou odpověď.

Druhá část modulu je Sniffer. Sniffer je psaný na míru toho, co se má měřit a je to často jen docela jednoduchá implementace jednoho rozhraní z Collectoru. Rozhraní má jednu metodu, ve které je vstupním parametrem konfigurace potřebná k měření a návratovým parametrem je list naměřených výsledků. Ve většině případů Sniffery vrací list s jedním měřením, v určitých specifických implementacích Snifferů se vrací několik měření najednou.

Na obrázku č. 2 je vidět strukturou nejsložitější Sniffer, který sbírá data z externího monitorovacího systému, ukládá si je do mezipaměti a Collector si pak data vyzvedává.



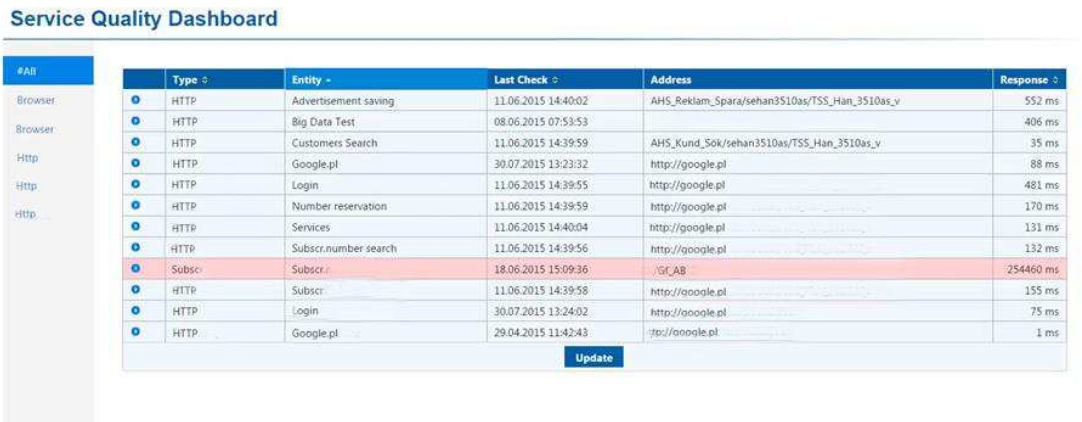
Obrázek 2 Vnitřní architektura kolektoru

Další ze Snifferů, které jsem vytvářel, byly:

- Http Sniffer – Měření odezvy HTTP GET požadavku na určité URL.
- Browser Sniffer – Vyrenderování webové stránky v reálném prohlížeči a změření jak dlouho trvá, než se stránka kompletně načte a zobrazí plus změření velikosti webové stránky v KB. Pro toto byl využit framework Selenium.

3.2.3 Dashboard

Dashboard je modul, který slouží k zobrazování měřených dat v reálném čase. První verze dashboaru vytvořená v systému mým kolegou byla za pomocí technologie JSF a jeho rozšíření Primefaces.



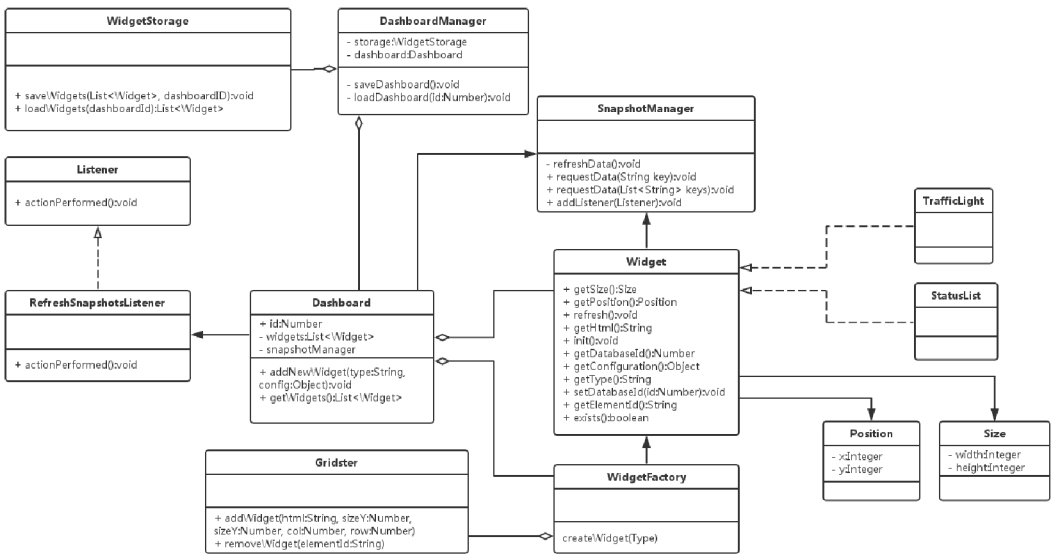
Obrázek 3 Starý dashboard

Funkcionalita této verze nebyla nijak veliká a i přesto každé další rozšíření znamenalo velkou investici času. U této technologie jsme zpozorovali, že lze velmi rychle vyvinout něco jako prototyp ale další dolaďování či dodělávání nestandartních věcí je časově velmi náročné a u mnoha věcí je člověk nucen říci, že to prostě v rozumném čase a bez hacků nepůjde. Navíc začaly být požadavky, aby si člověk mohl Dashboard nakonfigurovat a přidat si na něj Widgety.

Když nám teda kolega z projektu odešel, a já musel na tvorbě webu pokračovat, byla to hrůza. Proto jsem se rozhodl udělat Proof of Concept použití nové technologie místo stávajících JSF. Můj návrh byl použít stávající standarty HTML5 a CSS3 a JavaScript v jeho nové implementaci standartu ECMA Script 6. Dále použít templatovací jazyk Thymeleaf který je oproti JSF dobře podporovaný Springem. Thymeleaf měl být použit pro vyrenderování statických prvků na stránce a JavaScript měl zajistit všechno dynamické chování na stránce. S JavaScriptem bylo v plánu použít jQuery pro práci s DOMem a knihovnu GridsterJS pro vytvoření konfigurovatelných widgetů na Dashboardu. Dále použití frameworku MaterializeCSS pro pomoc se stylováním webu do Google Material Designu. Obrovskou výhodou by pak byla možnost vyvinou opravdu cokoliv si kdo zamane bez zbytečného se trápení.

Vyvinul jsem tedy prototyp Dashboardu s dvěma widgety. Jeden s obrázkem semaforu a druhý s listem měřených entit zobrazující stejná data jako původní dashboard. Prototyp mi byl schválen a začal tedy vývoj, na kterém se podílela i kolegyně, která se chtěla naučit práci v Java Scriptu.

Prvním krokem vývoje bylo vytvořit objektovou architekturu umožňující jedoduché vytváření nových widgetů. Každý widget přitom potřebuje rozdílná data ze serverů a dále je potřeba udržet všechny widgety datově synchronizované aby nedošlo ke zmatení uživatele. Je potřeba aby si uživatel mohl vytvořit svůj nový dashboard a ten si pak mohl uložit.



Obrázek 4 Architektura JS na klientské straně Dashboardu

Hlavní myšlenka tohoto návrhu je, že Widgety si zažádají u SnapshotManagera o data a on jeho úkolem je data periodicky získat ze serveru, ukládat je do společného Session Storage a oznamovat zpátky Widgetů, že jsou dostupná nová data. Widget pro provedení požadavku o data u Snapshot Managera používá jeden až několik klíčů, které jsou tvořeny částí URL adresy, kde jsou data dostupná na serveru. Stejný set klíčů Widget použije i pro už získání dat ze Session Storage, kde jsou data uskladněny.

Třída DashboardManager se stará o ukládání a načítání dashboardu s nastavenými widgety. Jedná se o serializaci konfigurace widgetů do databáze v JSON formátu. Při znovu sestavování již uloženého dashboardu se konfigurace předává zpátky Widgetům již při jejich inicializaci pomocí konstruktoru.

V návrhu jsem vytvořil Widget jako rozhraní a WifgetFactory, která se stará o dodávání instancí objektů implementujících rozhraní Widget. Factory je použita aby se v kódu zvýšila abstrakce vytváření instancí Widgetů, kód dashboardu je poté s využitím factory přehledný a krátký.

```
class Dashboard {
    constructor() {
        this._widgetFactory = new WidgetFactory();
        SnapshotManager.getInstance()
                       .addListener(new RefreshSnapshotListener(this));
        this._widgets = [];
    }

    addNewWidget(type, config) {
        const widget = this._widgetFactory.createWidget(type, config);
        this._widgets.push(widget);
    }

    getWidgets() {
        return this._widgets;
    }
}
```

Code 1 Dashboard implementace

Nicméně mít Widget interface by pěkně fungovalo například v jazyce Java, kde rozhraní jsou součást jazyka ale JavaScript, ve kterém je schéma naimplementované, rozhraní neobsahuje. Existuje ale několik způsobů, jak rozhraní v JavaScriptu emulovat za běhu programu. Tyto způsoby jsou převzaty z literatury [3] a přizpůsobeny našim potřebám a novějším standartům.

Známými způsoby jsou:

1. Popsání rozhraní pomocí komentáře
2. Emulace rozhraní pomocí ověřování atributů
3. Emulace rozhraní pomocí Duck Typing vzoru

Ad. 1 Popsání rozhraní pomocí komentáře

Nejjednodušší a nejméně efektivní metodou emulování rozhraní v JavaScriptu je pomocí komentářů. Jedná se o napodobování stylu jiných objektově orientovaných jazyků, klíčová slova *interface* a *implements* jsou použita, ale jsou zakomentována aby nezpůsobila syntaktickou chybu.

```
/*
 interface FormItem {
 function save();
 }
 */
const CompositeForm = function(id, method, action) { // implements FormItem
    ...
};
// Implement the FormItem interface.
CompositeForm.prototype.save = function() {
    ...
};
```

Code 2 Rozhraní pomocí komentáře

Tento způsob není nijak spolehlivý, neověřuje se, jestli třída opravdu implementuje nějaký set metod. Je to tedy spíše informativní způsob. Na druhou stranu nepřináší žádný výkonnostní overhead.

Ad. 2 Emulace rozhraní pomocí ověřování atributů

Druhá technika je trochu striktnější. Definice rozhraní stále existují jen v komentářích, ale třída explicitně deklaruje, která rozhraní implementuje. Funkce, která poté používá objekt, od které očekává, že bude implementovat nějaké rozhraní, může pomocí atributu ověřit, která rozhraní objekt říká, že implementuje.

Ad. 3 Emulace rozhraní pomocí Duck Typing vzoru

Podle zkušeností a jiných jazyků se dá říct, že na konci nám vlastně ani nezáleží, jestli třída deklaruje, že podporuje nějaké rozhraní, ale jestli obsahuje všechny vyžadované metody. A to je místo kde přichází Duck Typing vzor. Duck Typing bylo pojmenováno podle řečení „Pokud to chodí jako kachna a kváká jako kachna, je to kachna.“. Myšlenka za tímto postupem je jednoduchá. Pokud objekt obsahuje metody, které se jmenují stejně jako ty definované v rozhraní, tak objekt implementuje toto rozhraní. A za použití pomocné funkce můžete toto v metodě ověřit.

Tento přístup však vyžaduje opatrnější používání, jelikož přináší výpočetní overhead. Každé zkontrolování instance má v nejhorším případě kdy objekt neobsahuje správné metody složitost m*n, kde m je počet metod definovaných rozhraním a n je počet metod obsažených v kontrolované instanci. Situace se moc nelepší ani v případě, že instance obsahuje všechny metody definované v rozhraní. Zkontrolování velkého pole objektů by tedy mohlo být velmi náročné na zpracování.

Emulace rozhraní použitá v projektu

Způsob použitý na projektu je kombinací prvního a třetího způsobu. Jsou použity komentáře, abychom řekli, že nějaká třída implementuje rozhraní a pomocí pomocné metody můžeme kdekoliv ověřit, zda objekt, který nám přišel, obsahuje metody definované v rozhraní. Co se použití týče, ověřování je použito jen na místech, kde to je opravdu vyžadováno, jelikož ověření přináší operace Java Scriptu navíc.

```
export default class Interface {
    constructor(name, methods) {
        this.name = name;
        this.methods = [];
        for(var i = 0, len = methods.length; i < len; i++) {
            ... // Check if method names are strings
            this.methods.push(methods[i]);
        }
    }
    static ensureImplements(object) {
        ... // Check if min 2 arguments are passed
        for(let i = 1, len = arguments.length; i < len; i++) {
            const iface = arguments[i];
            for(let j = 0, mLen = iface.methods.length; j < mLen; j++) {
                const method = iface.methods[j];
                if(!object[method] || typeof object[method] !== 'function') {
                    throw new Error(`Interface.ensureImplements: checked object
                    does not implements the ${iface.name} interface.
                    Method ${method} was not found.`);
                }
            }
        }
    }
}
```

Code 3 Emulace rozhraní použitá v projektu

V implementaci schéma je tedy použito jen jedno rozhraní a to Widget, kde jsem chtěl mít na určitých místech opravdu kontrolu, zda objekt, který přišel do metody, opravdu obsahuje všechny metody, které má. V mnoha případech není v JavaScriptu vůbec potřeba rozhraní používat, což je dáno jeho obrovskou dynamičností.

Naimplementované Widgety

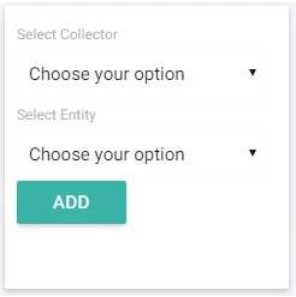
Pro vytvoření nového widgetu bylo potřeba implementovat metody definované v rozhraní, které vypadalo takto:

```
import Interface from './Interface'

const Widget = new Interface('Widget', ['getSize', 'getPosition', 'refresh',
'getHtml', 'init', 'getDatabaseId', 'getConfiguration', 'getType',
'setDatabaseId', 'getElementId', 'exists']);

export default Widget
```

Code 4 Widget interface

Postupně jsem naimplementoval 3 widgety a kolegyně poté naimplementovala další 2.

První widget byl *TrafficLightWidget*, který na začátku jen ukazoval barvou (zelená, žlutá, červená), jestli poslední měření bylo dobré, něco nebylo v pořádku, nebo dopadlo velmi špatně. A dále ukazoval, co měří a jaký byl poslední výsledek. Později jsem do widget přidal malý graf ukazující (hodnotou i barvou) historii několika posledních měření. Dále jsem dodělal zobrazení, kdy se entita naposled změřila.



Obrázek 5 Nastavení semaforu

Obrázek 6 Semafor

Druhý widget (StatusListWidget) měl kompletně nahradit starý dashboard. V první fázi obsahoval jen tabulku s typem měřené entity, jménem entity, data posledního měření, hodnotou posledního měření a barva řádku označuje, jestli bylo měření ve správných hodnotách, nebo nad povolené hodnoty. V druhé fázi přibyla možnost rozkliknutí řádku a zobrazení tabulky s historií, ve které si uživatel může zobrazit detaily jednotlivých záznamů z měření.

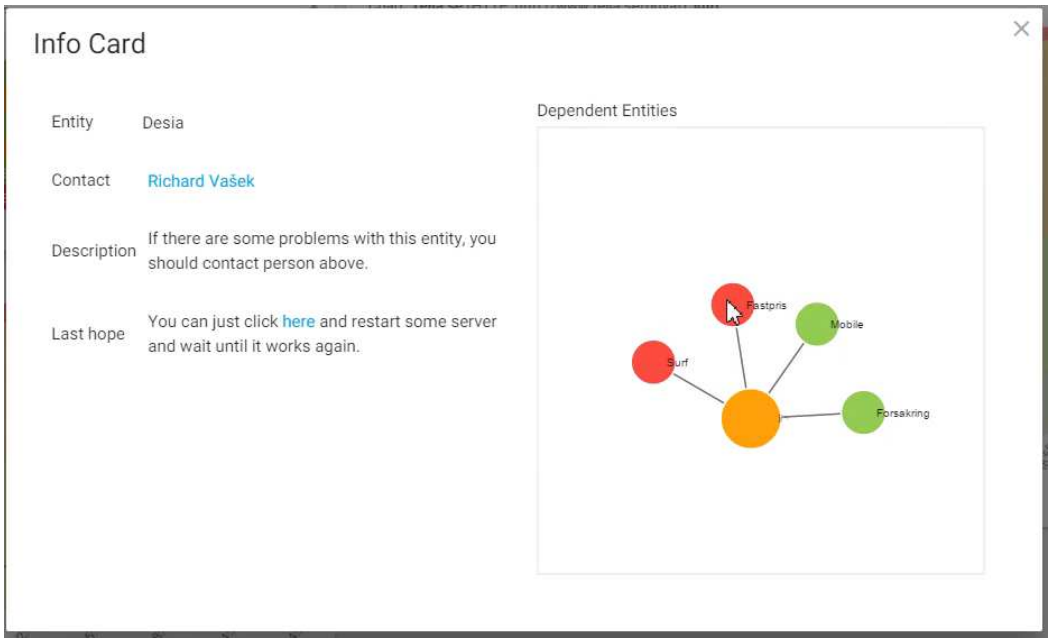
Obrázek 7 Status List

Třetím widgetem, byl odpočet do načtení nových dat na dashboard. Widget obsahuje canvas, do kterého jQuery plugin vykresluje odpočet. Dále je nastaveno načtení nových dat a restart odpočtu po kliknutí na widget.

Obrázek 8 Odpočet do obnovení dat

Informační karty – mockup

Zde šlo o vyvinutí karet zobrazujících závislosti mezi měřenými entitami a informace o entitách. Informace jsou určeny pro člověka sledujícího dashboard. Když nějaká entita přestane fungovat, informační karta má zobrazit kontakt na člověka, který je zodpovědný za entitu, popis co dělat a další informace vztažené k entitě. Dále je zobrazen graf entit, jejich aktuálního stavu a závislostí mezi nimi pro jednodušší dopátrání se, kde je vlastně problém.



Obrázek 9 Informační karta

Ač se jedná pouze o mockup data, začíná se zde objevovat špatná volba typu databáze konfigurátoru při založení projektu. Karty se od sebe dost liší tím, co obsahují a navíc se zobrazuje graf entit.

Pro vykreslení karet je použit ReactJS a pro vykreslení grafu knihovna D3.js

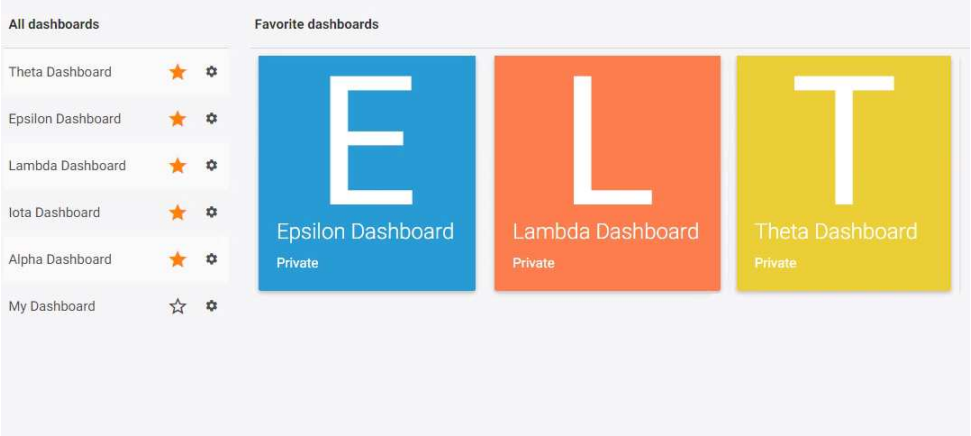
Sdílení dashbordů

Dalším požadavkem byla možnost vzít svůj vytvořený a nastavený dashboard a nasdílet jej jinému uživateli. Sdílet se může s 3 právy a to zobrazit, editovat a spravovat. Pouze s právem k zobrazení nemůže uživatel dashboard nijak editovat, právo editovat toto rozšiřuje o samotnou editaci a uložení dashboardu a správa dashboardu toto rozšiřuje o možnost dashboard sdílet dalším lidem a dashboard smazat.

Oblíbené dashboardy

S funkcionalitou sdílení dashboardů začal přicházet nový problém. Uživatel má v seznamu spousty dashboardů, a mnoho z nich ho většinu času nezajímají. Tento problém jsem vyřešil pomocí oblíbených dashboardů.

Přidání dashboardu do oblíbených je jednoduché. V seznamu všech dashboarů je možnost kliknout na hvězdičku a takto si dashboard přidat/odebrat z oblíbených. Oblíbené dashboardy poté zobrazuji jako karty na dvou místech v systému. První z nich je na hlavní stránce hned vedle seznamu všech dashboardů. Druhý z nich je přímo v dashboardu po kliknutí na ikonu v horním panelu.



Obrázek 10 Volba oblíbených dashboardů

Obrázek 11 Rychlá navigace na oblíbené dashboardy

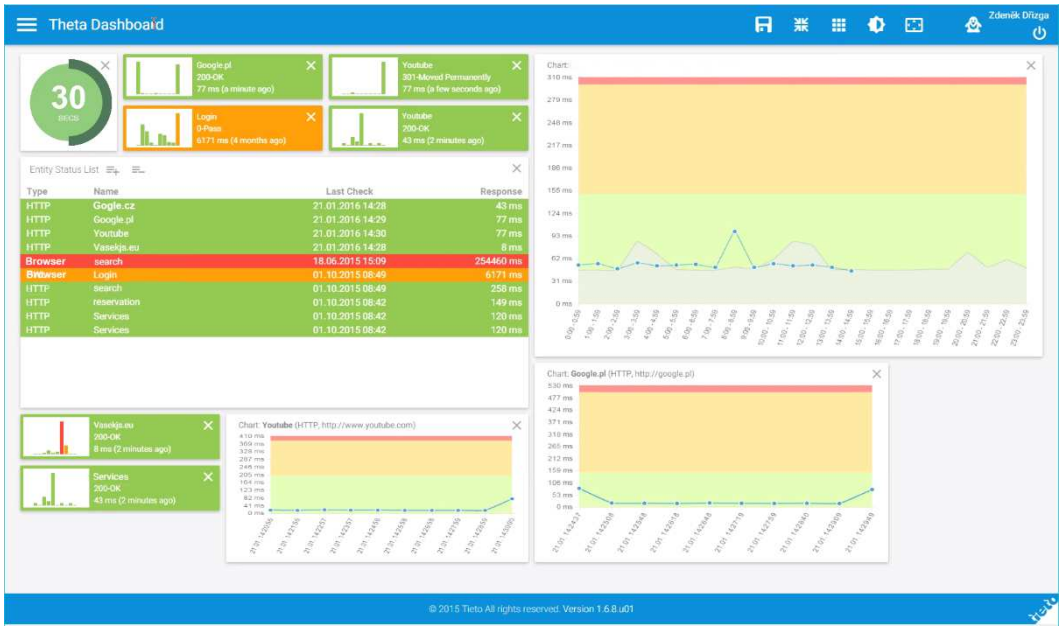
Noční režim

Dashboard obsahuje možnost manuálního přepnutí do nočního režimu. Noční režim jsem vyvinul z mé vlastní iniciativy. V podvečerních hodinách kdy na mě monitor s testovací verzí dashboardu příliš svítil a po delší době bylo nepříjemné se dívat na tak zářící monitor, jsem se rozhodl, vyvinou tuto funkcionalitu. Jedná se pouze o přidání CSS3 filteru na celý tag `<body>` html dokumentu a změna barvy pozadí dashboardu na černou. Zredukuje se takto obrovské množství bílé barvy, kterou dashboard obsahuje.

Zobrazení na celou obrazovku

Jedná se o použití knihovny screenfull.js pro přepnutí prohlížeče do režimu celé obrazovky pomocí tlačítka v html dokumentu.

Dashboard celkově



Obrázek 12 Celý nový dashboard

Klady

Přechod z JSF a Prime Faces rozšíření k Thymeleaf a JavaScriptu nám dovolil vytvořit opravdu to, co jsme chtěli. Oproti JSF jsme nebyli nijak omezováni a celý Dashboard je svižný.

Zápory

Status Engine modul byl navržen s REST API pro poskytování dat, tak jako v celém zbytku systému. Je to sice jednoduché řešení a ze začátku se zdálo být dobré, ale postupně se v celém systému vyvinul špatný vzor, kdy se klient v nějakém zvoleném intervalu buďto ptá, jestli jsou nová data, nebo si rovnou zažádá o data, která se od minulého dotazu nezměnily. Řešením by bylo zakomponovat do systému WebSocket technologii kdy server rovnou posílá nová data klientům, kteří se o tyto data zajímají.

3.2.4 Reporty

Cíl

Cílem bylo vytvořit dlouhodobé statistiky z měření. Uživatel by ze statistik mohl vyčíst, jak je měřená entita stabilní, jestli se její stabilita a rychlost zlepšuje/zhoršuje. Z reportů by také mělo jít vyčíst trendy (např. že server je o víkendu více vytěžovaný).

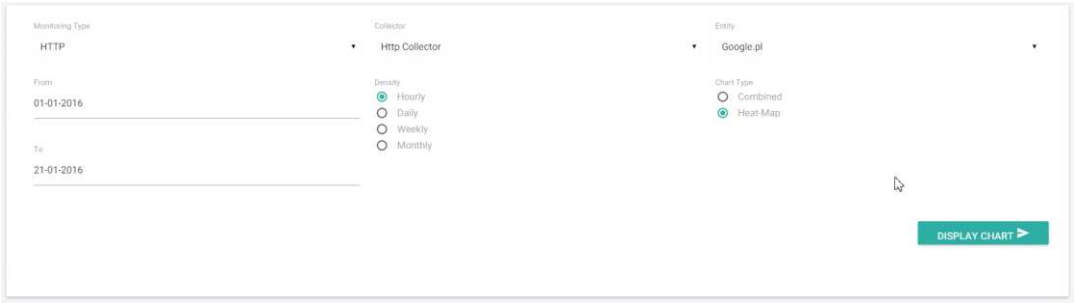
Backend část

Report Engine Modul se stará o poskytování dat z databáze. Obsahuje složitější SQL dotazy. Modul se nechává co nejvíce práce jako např. agregaci vykonat databázi, jelikož v tomto relační databáze celkem vynikají.

Frontend část

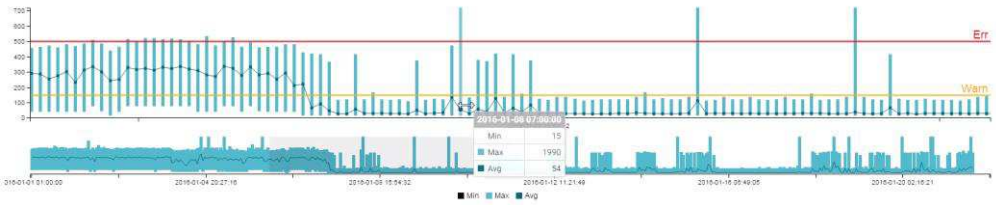
Webová část používá stejné technologie jako Dashboard. Web obsahuje formulář pro výběr entity, pro kterou chceme vytvořit report, dále nastavení vzhledu samostatného reportu. Na výběr je kombinovaný graf ukazující minimální, maximální a průměrné hodnoty měření a 2 prahy. Druhý typ grafu je inspirován teplotní mapou, kdy jednotlivé hodiny/dny obarvuje barvou od zelené až po červenou podle výsledku měření. Teplotní

mapa dovede zobrazovat minimální, maximální i průměrné naměřené hodnoty a dále počty vyskytnutých chyb.



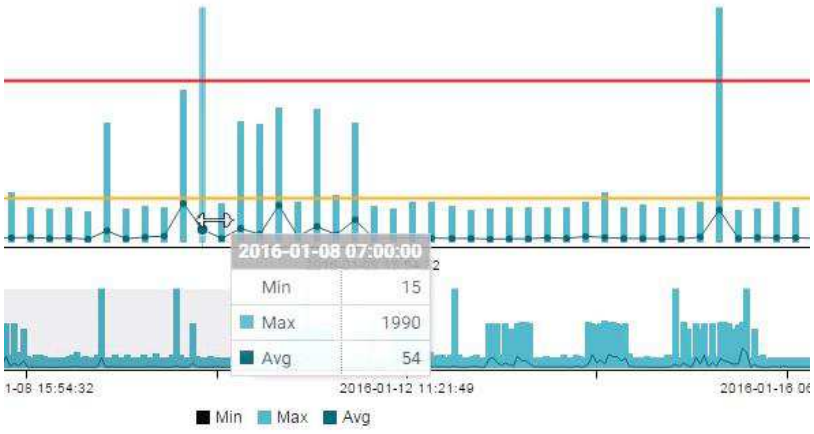
Obrázek 13 Nastavení reportu

Kombinovaný graf



Obrázek 14 Kombinovaný graf reportu

Kombinovaný graf (obr. 14 a 15) kombinuje sloupcový a čárový graf. Sloupcový graf ukazuje naměřené minimum a maximum v daném intervalu a čárový graf ukazuje průměr měření v daném intervalu. Dále jsou žlutou a červenou čarou zobrazeny nastavené prahy měření pro danou měřenou entitu. Graf je vytvořen za pomocí knihovny D3JS, která jej vykresluje jako vektor pomocí svg elementů. Graf je interaktivní, dá se v něm přibližovat a pohybovat.



Obrázek 15 Detail kombinovaného grafu

Veškeré zpracování dat pro graf probíhá přímo v databázi pro vysokou efektivitu práce nad velkým množstvím dat, které se pohybuje často i v milionu naměřených výsledků. Graf na klientské straně (obr. 14 a 15) už tedy zpracovává data v maximálně desítkách KB.

Teplotní mapa

Druhým dlouhodobým grafem je tabulka inspirovaná teplotními mapami. Řádky ukazují jednotlivé dny/týdny a sloupce ukazují jednotlivé hodiny/dny. Naměřené hodnoty se porovnají s nastavenými prahy měření a určí se barva buňky. Určení barvy není lineární, což vychází z algoritmu, který vezme minimální a maximální hodnotu v celé tabulce plus 2 prahy nastavené na entitě, čímž vzniknou 4 prahy, tzn. 3 intervaly, do kterých se rozdělí barvy po 3, a to 3 pro OK, 3 pro Warning a 3 pro Error. Tímto vznikne 9 intervalů kde každému je přiřazená jiná barva. Každá hodnota v tabulce je pak přiřazena do jednoho z 9 intervalů a tím je jí určena barva.

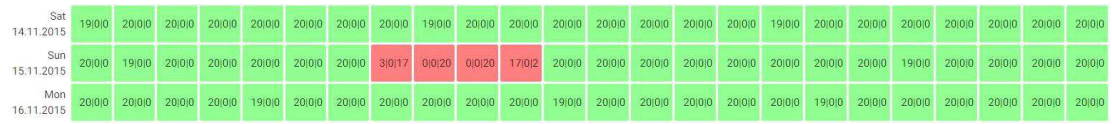
```
getColor(value: number):string {
    if(value < this.thresholdWarning) {
        const minValue = this.minimums[this.dataType];
        const diff = (this.thresholdWarning-minValue)/this.OK_COLORS.length;
        const index = Math.floor(value/diff);
        return this.OK_COLORS[index];
    }
    else if(value < this.thresholdError) {
        const diff = (this.thresholdError-this.thresholdWarning)
            /this.WARNING_COLORS.length;
        const index = Math.floor((value-this.thresholdWarning)/diff);
        return this.WARNING_COLORS[index];
    }
    else {
        const maxValue = this.maximums[this.dataType];
        const diff = (maxValue-this.thresholdError)/this.ERROR_COLORS.length;
        const index = Math.floor((value-this.thresholdError)/diff);
        return this.ERROR_COLORS[index];
    }
}
```

Code 5 Vyhodnocení barvy

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Fri 01.01.2016	317	336	322	328	317	329	329	310	329	339	327	323	320	336	306	324	323	325	313	256	325	327	326	311
Sat 02.01.2016	326	309	309	320	316	328	328	274	319	246	321	326	326	319	323	327	315	312	302	276	327	339	311	329
Sun 03.01.2016	322	312	321	336	320	321	334	315	329	329	318	322	325	314	324	305	311	277	333	312	326	321	333	339
Mon 04.01.2016	326	333	210	244	292	228	277	234	196	158	201	249	290	255	292	194	331	394	257	328	303	320	321	338
Tue 05.01.2016	314	318	258	316	278	314	302	258	302	283	275	290	283	258	304	200	228	304	328	328	308	318	323	304
Wed 06.01.2016	320	329	252	238	279	290	286	253	274	301	231	311	333	299	242	250	328	315	322	311	330	321	335	318
Thu 07.01.2016	308	280	269	336	323	276	332	280	290	251	292	210	220	65	91	46	26	26	56	29	29	30	27	28
Fri 08.01.2016	25	50	26	31	34	132	54	27	57	39	126	33	63	38	83	26	29	28	28	27	27	25	26	26
Sat 09.01.2016	29	27	33	30	27	27	26	26	31	112	27	30	29	29	27	29	27	28	27	29	28	30	29	30

Obrázek 16 Teplotní mapa měřených časů

Krom naměřených hodnot dokáže mapa (obr.16) zobrazit i naměřené chyby (obr. 17). Zde je algoritmus pro zobrazení daleko jednodušší. Pokud jsou v časovém intervalu všechny maření v pořádku, políčko je zelené, pokud je alespoň jeden warning, políčko je oranžové a pokud alespoň jedna chyba, políčko je červené.



Obrázek 17 Teplotní mapa chyb

Toto zobrazení se ukázalo jako velmi užitečné, jelikož mapa zobrazuje opravdové chyby na rozdíl od mapy na obrázku 16, kde pokud se změřila chyba, ale měření proběhlo dostatečně rychle, nikdo nepozná, že bylo něco špatně.

3.2.5 Autentizace uživatelů

Pro autentizaci uživatelů v systému je vyčleněn Maven modul v konfigurátoru. Modul je vytvořen tak aby se dal vyměnit za jiný. Standartně se používá pro autentizace LDAP a pro prostředí, kde není LDAP dostupný se používá kolekce staticky definovaných uživatelů.

Proto jsem tedy vytvořil 2 moduly, které se dají přepínat při kompilaci konfigurátoru, který pak poskytuje REST API pro přihlášení uživatele do systému.

3.3 Vedení projektů

Druhým větším úkolem, který mi byl dán, bylo převzetí, údržba a rozvoj postupně celkem šesti projektů. Většina těchto projektů už byla funkční a běžící v produkci. K dispozici jsem na začátku dostal 8 dalších stážistů, kteří měli na projektech pracovat. Projekty jsem začal přebírat postupně jeden po druhém, abych byl schopen najít na projektu práci a rozdělit ji mezi stážisty.

Jira

Pro rozdělování práce jsem používal systém Jira, který má v Tietu velkou oblibu. Pro jednoduchost jsme začali používat Kanban.

3.3.1 Systém na rezervaci sdílených míst

První systém, který jsem přebral, byl systém pro rezervaci sdílených míst. Systém již běžel v produkci a umožňoval si v budově zarezervovat sdílené místo na celý den. Systému se v podstatě nikdo přibližně půl roku nedotkl a začaly se na něm hromadit nahlášené chyby a začaly přicházet požadavky na nové funkcionality systému.

Systém byl napsaný jako monolit za pomocí J2EE technologií. Projekt neměl žádnou pořádnou strukturu, všechno bylo promíchané, obsahoval kupy nepoužívaného kódu a kupy nepoužívaných sloupců v databázi. Navíc přišel požadavek na vytvoření mobilní aplikace a REST rozhraní pro tento systém. Dále možnost si zarezervovat jen několik hodiny místo celého dne.

Z J2EE do Spring Bootu

Z mého prvního pohledu kdyby na systému začalo najednou pracovat několik lidí, které ani nemají žádnou zkušenost s vývojem v J2EE technologii, projekt by se zhroutil dříve než by začal. Takže se nabízela myšlenka projekt zrefaktorovat. Požadavek na nové funkcionality stejně vyžadoval dost měnit strukturu databáze.

Rozhodl jsem se tady projekt založit znova a inspirovat se původním návrhem systému. Vzal jsem existující rozhraní ze systému, abych věděl, jaké metody vůbec obsahuje. Užitečné jsem refaktoroval, ty které se zdály nepoužívané, jsem označil jako @Deprecated. Vytvořil jsem tak rozhraní pro service vrstvu.

Dále jsem vytvořil i nové modely pro service vrstvu, které jsou flexibilnější než původní a jsou v nich obsaženy nové položky potřebné k nové funkcionalitě. Dalším krokem bylo vytvořit rozhraní pro přístup k databázi.

Pro přístup k relační databázi jsem použil repository vzor. Spring JPA nabízí možnost rozšířit jejich CRUDRepository rozhraní a získat vygenerovanou implementaci metod vytvořených v rozhraní při běhu aplikace. Spring JPA nabízí 3 možností na tvoření dotazů pro databázi. První z nich (vhodný pro jednodušší dotazy) generuje implementaci podle jména metody. Příklad takové ho názvu metody je

```
List<Person> findAllByFirstNameAndLastName(String firstName, String lastName)
```

Code 6 Ukázka psaní query pomocí názvu metody

Druhým způsobem je anotace @Query, která nabízí možnost použít JPQL a nativní dotazy. Třetím způsobem je použití Entity Managera. V projektu jsem použil kombinaci prvních dvou způsobů.

Dále jsem vytvořil implementaci service rozhraní. Tato implementace obsahuje business logiku, která byla v původní verzi často obsažena v ManagedBeanách od JSF, což bylo velmi špatné řešení.

Dalším krokem bylo vzít věci patřící JSF ze starého projektu a přesunout je do nového a napojit na nová rozhraní. Po zdlouhavém hledání chyb a napojení je projekt zase funkční a rozdělen do 3 vrstev. První vrstva je view v podobě JSF nebo REST API pro mobilní aplikaci. Druhou vrstvou je service vrstva a třetí vrstvou jsou přístupy k datům. JPA pro relační databázi a LDAP pro data týkající se uživatelů. Service vrstva tedy pracuje se dvěma moduly pro přístup k datům a data spojuje dohromady pomocí logiky.

Pro zabezpečení REST API jsem vytvořil 2 moduly používající OAuth2 strategii. Oba moduly využívají plně Spring Boot pro jejich Convention over Configuration, čímž jsem si ušetřil práci s konfigurováním komunikace mezi těmito dvěma moduly. Prvním modulem je OAuthServer, tento modul slouží pro přihlášení uživatelů a poskytnutí tokenu. Jeho druhou prací je na základě tokenu zpětně poskytnout informace o uživateli. Druhým modulem je OAuthResource, tento modul slouží pro zabezpečení REST API. Klient musí poslat token, který dostal od prvního modulu a druhý modul nám zařídí, že na základě tohoto tokenu zjistí od prvního modulu, o koho se jedná a předá nám v Rest Endpointu objekt obsahující informace o uživateli, který vytvořil dotaz.

Z aplikačního serveru do Tomcatu

Původní aplikace běžela na Java verze 7 a Aplikačním serveru Jboss, který nepodporoval novější Javu. Pro Spring tyto aplikační servery obsahují spousty věcí, které by zůstaly nevyužity. Rozhodl jsem se tedy, že zvolím menší a lehčí server a to Tomcat ve své aktuálně nejnovější stabilní verzi 8. Tomcat je ve Spring komunitě velmi používaný a Spring Framework je pro něj dobře optimalizovaný.

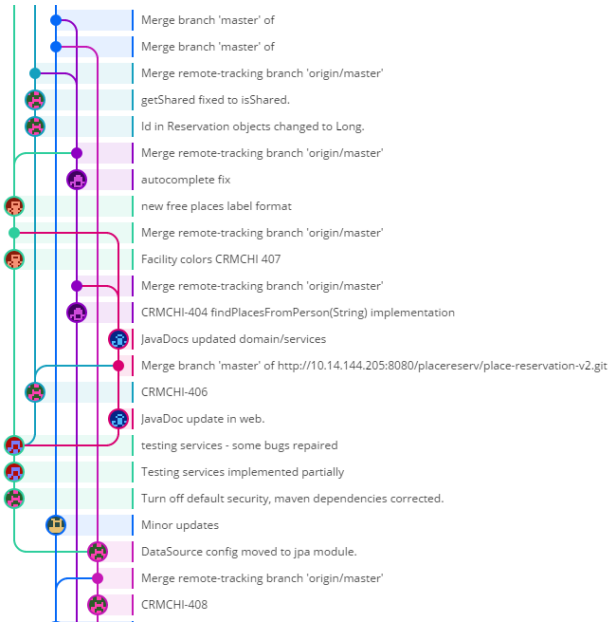
Problém který jsem tedy musel vyřešit, byla změna závislostí projektu. Původní Jboss v sobě obsahuje velký počet knihoven, které poskytuje pro aplikaci. Tomcat naprostou většinu těchto knihoven neobsahuje, tudíž se musí všechny knihovny připojit ke zkompilované aplikaci pomocí Mavenu. Dále jsem musel vyřešit správné logování aplikace. Zvolil jsem pro logování knihovnu Log4j a nakonfiguroval formát a typy výpisů do logů. Dále jsem nainstaloval a nastavil Tomcat na Linux serveru a nastavil Jenkins, tak aby byl schopen kompilovat aplikaci a nahrávat ji na server automatizovaně.

GIT rebase strategie

Prvním krokem pro týmový vývoj na projektu bylo zavedení verzovacího systému. Dnes nejpoužívanějším je GIT a vzhledem k tomu, že už jsem s ním měl dobrou zkušenost, rozhodl jsem se pro jeho použití. Zařídil jsem tedy školení pro stážisty, kteří jej neznali. Prvních 2 týdny používání GITu po školení, jsem došel k závěru, že budu muset vymyslet a nastolit nějaký způsob práce s GITem. V historii GITu se totiž nedalo nic pořádně dohledat.

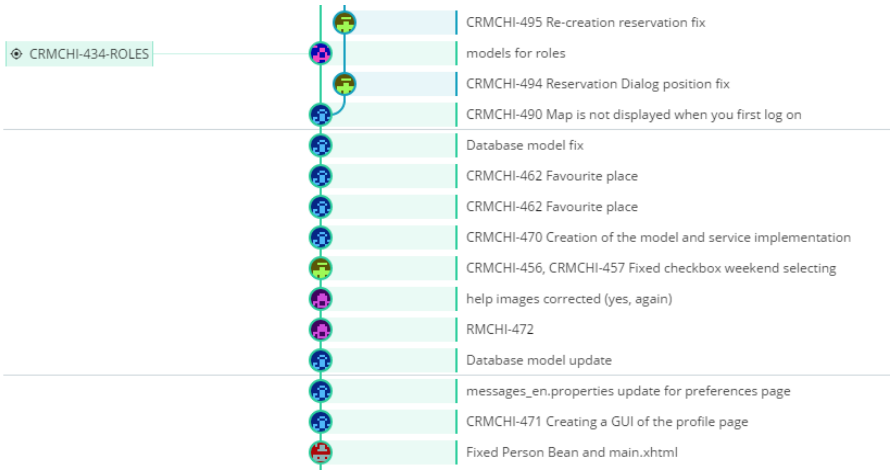
Rozhodl jsem se použít Rebase Strategii společně s tvořením tzv. Squash commitů. Myšlenkou je udržet hlavní větev přímou a kupy malých commitů squashnout 1 až 2 commitů na jeden úkol. Je tady zaveden zákaz tvorby non-fast-forward mergů do hlavní větve. Další věcí bylo zavedení psaní rozumných komentů u commitů. Volba padla na zprávu, která obsahuje po sobě na jednotlivých řádcích ID úkolu z Jira systému, popis úkolu z Jira systému a popis provedené práce.

Na zavedení této strategie jsem musel, udělal školení, ve kterém jsem předvedl a vysvětlil tento GIT workflow, jelikož stážisti nebyli schopni s GITem pořádně pracovat. Nakonec se ukázalo, že největším problémem pro ně je řešení konfliktů. Workflow s GITem se celkem rychle zlepšil a začal být přehlednější. Pro srovnání první obrázek ukazuje naši práci pomocí merge strategie a druhý obrázek ukazuje naši práci týden po zavedení rebase strategie, kdy všichni začali postupně dodržovat pravidla. Graf poskytnutý GITem je hned přehlednější a jde jednoduše vidět kdo, co udělal a změnil bez dlouhého prohledávání grafu.



Obrázek 18 Merge only strategie

Jak jde z obrázku č. 18 vidět, historie není přehledná a obsahuje mnoho merge commitů, které nic neříkají. Cílem tedy bylo zpřehlednit tento stav.



Obrázek 19 Rebase strategie

Zde na obrázku č. 19 už jde jasně vidět, jak se graf zlepšil a zpřehlednil. Dalším mým krokem pak bylo tyto pravidla přesně zadefinovat a sepsat návod pro přisívání do projektového repozitáře, který obsahuje i pravidla pro vhodný tvar commit message.

Automatizace nasazení na test server

Pro automatizované nasazování aplikace na testovací server je použit systém Jenkins, který při každém provedení push do hlavní větve se pokusí projekt zkompilovat a pokud je úspěšný, nahraje aplikaci na testovací server.

JavaScript školení

Ačkoliv JSF se snaží omezit, aby člověk musel používat JavaScript, stejně se tomu nikdo nevyhne. Problém je, že spousta lidí svět JavaScriptu nesleduje, takže neví o nových standartech a stále vyvíjí ve sterých verzích. Rozhodl jsem se tedy vytvořit celodenní školení JavaScriptu pro stážisty. Školení začínalo popisem ECMAScript standartu a základech v ES5. Poté základy ES6 a jejich novinek. Nakonec jsme si společně v ES6 nastavili transpilátor a vyvinuli vlastní jednoduchý framework pro vývoj webových aplikací inspirovaný myšlenkou ReactJS a Redux architekturou.

Spring boot admin

Hlavní myšlenkou tohoto úkolu bylo poskytnout informace o našich aplikacích v reálném čase. Chtěli jsme tedy zobrazit na monitoru důležité metriky našich aplikací, jako je velikost JVM haldy, jestli aplikace běží, jestli funguje přístup k databázi a jestli obsahuje nějaké chyby.

Jelikož používáme pro naše aplikace Spring, máme možnost použít projekt Actuator od Springu. Actuator po přidání do projektu poskytne veřejně dostupné REST API, které poskytuje spousty důležitých informací o aplikaci. Druhým krokem bylo toto REST API využít, proto jsem našel projekt Spring Boot Admin. Tento projekt se skládá z několika modulů, jeden z nich jsem použil jako závislost v naší aplikaci a nastavil v něm, na jaké URL bude dostupný administrativní modul. Další z modulů jsem zkompiloval a nasadil na server. Naše aplikace se tedy po startu podívá na zadanou URL a přihlásí se k admin modulu, který je pak schopen použít data z našeho REST API a zobrazit je pomocí webového rozhraní.

Jira

Jira je jedním z největších a nejrozšířenějších zástupců systémů pro správu úkolů projektech. Atlassian, společnost, která stojí za vznikem tohoto systému, dále nabízí další produkty, z nichž, krom Jiry, nejpoužívanějšími jsou Confluence a BitBucket.

Jiru jsme používali jak na projektu pro monitorovací systém Karellen, kdy její využívání bylo pro nás bylo spíše otravné než přínosné. Tak později na dalších projektech již s větším týmem vývojářů, kdy se tento systém projevil jako velmi užitečným a velmi potřebným pro rozdělování práce.

Tento popsaný rozdíl mezi neužitečností a užitečností tohoto systému spočívá tedy na počtu členů týmu. Na prvním projektu jsem pracoval se dvěma kolegy, poté nějakou dobu sám a poté s jednou kolegyní, tudíž se nám takto obrovský a rozsáhlý systém zdál trochu nepotřebný. Na dalších projektech, kdy počet vývojářů vstoupl postupně na 12 vývojářů, začal být systém opravdu potřebný, aby 2 lidé nedělali stejnou práci a abych já mohl dopředu naplánovat práci pro tým.

Confluence

Confluence je jeden z dalších produktů firmy Atlassian. Slouží pro poskytnutí prostoru pro skladování informací, znalostí, dokumentů a souborů o projektech. Slouží v podstatě jako projektová wiki, která je synchronizovaná s projektovou Jirou.

Další úkoly

Krom přepsání systému do nové rozdělené architektury a do Springu místo J2EE byly další mé vývojářské úkoly zaměřené hlavně na opravy chyb v systému a pak taky další refaktorování.

Po nějaké době vývoje jsem opět refaktoroval celou strukturu databáze abych uklidil po ostatních členech týmu. Když už používáme relační databázi, chtěl jsem, abychom správně využili cizí klíče místo řetězce znaků a indexy pro zrychlení vyhledávání v databázi. Dále jsem refaktoroval některé špatné dotazy na databázi. Tyto změny jsem musel dodat i na servisní vrstvu aplikace a vše znovu otestovat.

Dalším úkolem bylo odstranění LADP jako částečného zdroje dat z našeho systému. Po delším vývoji aplikace kdy jsme místa, kde sedí zaměstnanci Tieta brali z externího LADP systému a také po zjištění, jak dlouho trvá vyřizování čehokoliv, co se týká LDAPu s Tietí podporou, jsme se rozhodli, že budeme ukládat místa lidí v naší vlastní relační databázi a LDAP použijeme jen pro prvotní naplnění. Toto řešení umožní lepší manipulaci s těmito daty, jelikož projekt má do LDAP práva pouze pro čtení, nikoliv editaci údajů.

Toto rozhodnutí pro mě znamenalo lehkou úpravu databáze a dále úpravu vnitřní logiky aplikace, která se díky tomuto také zjednodušila. Posledním krokem pro mě byla migrace stávajících dat z LDAP do naší databáze.

Zhodnocení mé práce na projektu

Tento projekt mi poskytl možnost vyzkoušet si vedení lidí na projektu, čímž jsem získal obrovskou zkušenost, kterou bych s těží sháněl ve školní sféře jako student. Dále jsem získal zkušenost z přebírání projektu po jiných programátorech a z této zkušenosti jsem se odnesl obrovské ponaučení, jak důležité je psát dokumentaci k projektu a udržovat ji aktuální a dále jak podstatné je zvolit správnou architekturu pro určitý projekt. Právě chybějící dokumentace mě dovedla k tomu, abych založil Wiki na Confluence o našem projektu, kde jsem vkládal své vědomosti o tomto projektu a dále mě dovedla k psaní správného JavaDocu.

Bohužel jsem si z projektu odnesl i špatné zkušenosti. První z nich se týká technologie Hibernate, která tvoří objektově relační mapování mezi aplikací a relační databází. Setkal jsem se s mnohými problémy s touto technologií a to hlavně z výkonnostního hlediska. Druhá stránka tého technologie je, že jsem společně s JPA vytvořit prototyp aplikace velice rychle. Další špatná zkušenost je opět s technologií JSF. Tato technologie přiměla nepříliš zkušené vývojáře psát špatný kód na špatná místa a opět její rychlost se projevuje jako špatná. Špatnou zkušenost mám taky s úpravami již vytvořených komponent jak z JSF tak z rozšíření PrimeFaces. Druhá stránka této technologie je opět možnost rychlého vývoje jednoduchých aplikací.

Mými dalšími plány na projektu je provést optimalizace Hibernate, popřípadě i zvážit přechod na jiný druh databáze, který by byl přívětivější pro vývojáře a vhodnější pro aplikaci. Dále bych se chtěl zbavit technologie JSF a nahradit jí REST API vrstvou s možnou kombinací s WebSockets technologií a vytvořit kompletně oddělené webové rozhraní v některé z JavaScript technologií například ReactJS s Redux architekturou. Tímto bych taky umožnil fontend vývojářům se soustředit na frontend bez toho aby se museli přehrabovat kupou Java kódu, jak je tomu doteď v JSF.

3.3.1 Digitalizace formulářů

Dalším projektem, který jsem přebral, byl projekt pro digitalizaci papírových formulářů pro nové zaměstnance Tieta. Projekt už byl ve fázi odlaďování a dodělávání posledních požadavků. Projekt tedy prošel dvakrát přes review aby se dodělaly vývojářské dluhy. Na projektu tedy zůstal jeden stážista, který jen dokončil.

Bohužel u review se odhalil dost špatný návrh celého systému. Systém není vůbec flexibilní. Vytvoření druhého lišícího se formuláře by zabralo spousty práce. Problém spočívá v návrhu doménového modelu a volbě relační databáze. V podstatě pro každý vstupní box na webu existuje objekt, takže pro vyplnění jednoho formuláře zde existuje zhruba 20 doménových modelů. A k tomu podobné množství tabulek v databázi.

Jelikož už se vývoj tohoto systému ukončuje, nechal jsem návrh tak jak je. Jestli bude vývoj někdy pokračovat, bylo by dobré zvážit, zda celý systém nepřepsat. První věcí by bylo použití NOSQL databáze pro ukládání formulářů. Použití dokumentové databáze jako je například MongoDB by bylo mnohem lepší volbou pro svou flexibilitu a schemaless. Další věcí by bylo přepsání doménových modelů do několika univerzálních modelů, které by se daly použít pro všechny formuláře. Třetí věcí by mohlo být nahrazení JSF technologie za REST API a vytvoření klientské části v JavaScript FW jako je Angular 2 nebo ReactJS. Velká část Javy by se mohla odstranit a bylo vy jasněji vidět co je klientská část a co je serverová část. Dále by bylo webové gui i mnohem svižnější a nemělo by designové problémy JSF.

4 Uplatněné a chybějící znalosti

4.1. Znalosti ze školy uplatněné na praxi

Na praxi jsem uplatnil některé ze znalostí nabytých ve škole. Pro samotné programování pro mě byly velmi přínosné znalosti z předmětů Algoritmy II a Programování II. Dále pro navrhování architektury pro mě byly velmi přínosné zkušenosti z předmětu Úvod do Softwarového Inženýrství, kde jsem se naučil tvorbu UML a základní návrhové vzory a dále pro mě byly velmi užitečné enterprise návrhové vzory, které jsme se učili v předmětu Vývoj Informačních Systémů.

Další předměty, které chci zmínit je Úvod do Databázových Systémů a Databáze a Informační Systémy, které mi daly mnoho informací o relačních databázích, které jsem následně využil při návrhu relačních schémat databází.

4.2. Chybějící znalosti

I přes mnoho znalostí, které jsem si sebou na praxi donesl ze školy, jsem se musel mnoho věcí na praxi doučit.

Jednou z těchto znalostí je verzovací systém GIT, který jsem před svou praxí neznal. Z mých pozorování je GIT je momentálně nejpoužívanější systém na sdílení i zálohování a kódu proto bych ocenil, kdyby tento systém byl součástí některého z předmětů například již ve druhém ročníku.

Další technologií jsou NOSQL databáze, o kterých jsem na škole nic neslyšel. Zpětně je to pro mě největším zaznamenaným problémem, kdy kvůli neznalosti této technologie jsme u jednoho z projektů na jeho začátku špatně zvolili relační databázi a pak se trápili s následky. Takže bych někdy ve druhém ročníku uvítal předmět o NOSQL databázích.

5 Závěr

Na své odborné praxi jsem vytvořil monitorovací systém, který dokáže monitorovat různé externí systémy. Hlavní částí je monitorování pomocí jednoduchých HTTP GET požadavků, druhou a složitější je monitoring, pomocí simulace uživatele v prohlížeči. Dále jsem si zkusil vést tým vývojářů a zároveň s tímto týmem vyvíjet aplikaci pro rezervaci míst. V průběhu své praxe jsem se naučil pro mě nové technologie a získal zkušenosti jak z vývoje software v týmu, tak z vedení týmu vývojářů. Dále jsem získal mnoho zkušeností z návrhu architektury moderních webových systémů a poznal jsem, jak důležitá je komunikace v týmu. Na konec svou praxi hodnotím velmi pozitivně, dala mi možnost nahlédnout do světa práce vývojáře na projektech a boje s termíny. Získal jsem tedy mnoho nových znalostí a zkušeností v docela krátkém čase a míním ve firmě nadále pokračovat na stáži.

Literatura

[1] GAMMA, Erich. *Design patterns elements of reusable object-oriented software*. Reading: Addison-Wesley, c1995. Addison-Wesley professional computing series. ISBN 0-201-63361

[2] FOWLER, Martin. *Patterns of enterprise application architecture*. Boston: Addison-Wesley, c2003. Addison-Wesley signature series. ISBN 0-321-12742-0

[3] HARMES, Ross. a Dustin. DIAZ. *Pro JavaScript design patterns*. New York, NY: Distributed to the book trade worldwide by Springer-Verlag New York, c2008. ISBN 159059908X

[4] *Tieto - IT, výzkum a vývoj a poradenství.* [online]. Ostrava: Tieto Czech, 2016 [cit. 2016-04-18]. Dostupné z: https://tieto.cz

[5] *Spring Boot Reference Guide* [online]. Los Angeles: Pivotal, 2016 [cit. 2016-04-18]. Dostupné z: http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/